CMPSC 465 Data Structures & Algorithms
Spring 2024 Mehrdad Mahdavi and David Koslicki

HW 7

1. (20 pts.) **Dijkstra's Algorithm.**

Solution:

(a) The order of vertices in which they are removed from the priority queue is s, a, c, d, b.

(b) As the order of vertices being removed from the priority queue needs to be same as before, we need to guarantee $distance(s,c) < distance(s,d) < distance(s,b)$. Changing $l(a,d)$ only potentially affects $distance(s,d)$ and is not possible to affect $distance(s,c)$ as well as $distance(s,b)$. So, we are certain that $distance(s,c) = 2$ and $distance(s,b) = 7$. Thus, our goal is to make sure $2 < distance(s,d) < 7$.

$$distance(s,d) = \begin{cases} l(s,a)+l(a,d) = 1+l(a,d) & \text{if } 1+l(a,d) \le 5 \\ l(s,a)+l(a,c)+l(c,d) = 5 & \text{if } 5 \le 1+l(a,d) \end{cases}$$

With the consideration of $2 < distance(s,d) < 7$, the condition of the first case ($(s,a) \to (a,d)$ is the shortest path from s to d) is $2 < 1+l(a,d) \le 5$, and the the condition of the second case ($(s,a) \to (a,c) \to (c,d)$ is the shortest path from s to d) is still $5 \le 1+l(a,d)$. Overall, as we consider both cases, the range of $l(a,d)$ is $(1, \infty)$.

2. (20 pts.) **Roads and planes.**

Solution: We first recognize the SCC of our graph treating roads as bi-directional edges. If two nodes are connected by roads, then they must be in the same SCC, and the additional constraint about planes ensures that no plane edge will be inside a SCC. Thus the edges across different SCCs are exactly the plane routes.

We can find the SCCs using the SCC algorithm in linear time. Additionally, we can identify major cities by using DFS in linear time, for example. Within the SCC containing s, we can run Dijkstra's to find the shortest path from s to all other nodes in the same SCC. We can do the same for all other SCCs from their major city. Now, we can reconstruct the graph, replacing SCCs that have more than 1 node. For each such SCC, leave the major city in place, but remove all other nodes that do not have planes leaving from them, and give an edge of cost C connecting the major city to each of these nodes. Now, since all edges have cost C, run shortest-path BFS from s to find the shortest path from s to all other remaining vertices. Pairing this information together gives the shortest path from s to all other reachable nodes. This is done by summing the costs of the plane edges and the cost of the shortest path in each SCC returned by Dijkstra's to get from a major city to the node connecting to a plane edge along the path. For other cities, BFS gives the shortest path to a major city, and Dijkstra's gives the shortest path from such a major city to any other city reachable using only roads. This gives the shortest path since it is not possible to reach such a city without first going through the associated major city. The running time will be linear since Dijkstra's can run in $O(|V|^2)$, but since the total number of nodes Dijksta's visits is $(O\sqrt{|V|})$, it runs in $O(|V|)$. The algorithm works since all plane edges and the shortest paths within each SCC have the same cost, so in the metagraph, all edges have the same weight, allowing for finding the shortest path with BFS. Additionally, since all plane edges go to a major city, it is not possible to arrive at a SCC without going through the major city, so the only path that goes through that SCC starts at the major city, allowing Dijkstra's to be run from the major cities (and s) only.

3. (20 pts.) **Start Node Negative Edges.**

Solution: Dijkstra's algorithm would work.

Proof: Consider the proof of Dijkstra's algorithm. The proof depended on the fact that if we know the shortest paths for a subset $S \subseteq V$ of vertices, and if (u,v) is an edge going out of S such that v has the minimum estimate of distance from s among the vertices in $V \backslash S$, then the shortest path to v consists of the (known) path to u and the edge (u,v). We can argue that this still holds even if the edges going out of the vertex s are allowed to be negative. Let (u,v) be the edge out of S as described above. For the sake of contradiction, assume that the path claimed above is not the shortest path to v. Then there must be some other path from s to v which is shorter. Since $s \in S$ and $v \notin S$, there must be some edge (i,j) in this path such that $i \in S$ and $j \notin S$. But then, the distance from s to j along this path must be greater than the estimate of v, since v had the minimum estimate. Also, the edges on the path between j and v must all have non-negative weights since the only negative edges are the ones out of s. Hence, the distance along this path from s to v must be greater than the estimate of v, which leads to a contradiction.

4. (20 pts.) **Shortest Path in Currency Trading.**

Solution:

a) Represent the currencies as the vertex set V of a complete directed graph G. To find the most advantageous ways to converts c_s into c_t, you need to find the path $c_{i_1}, c_{i_2}, \cdots, c_{i_k}$ maximizing the product $r_{i_1,i_2} r_{i_2,i_3} \cdot \cdots \cdot r_{i_{k-1},i_k}$. This is equivalent to minimizing the sum $\sum_{j=1}^{k-1}(-\log r_{i_j,i_{j+1}})$. Hence, it is sufficient to find a shortest path in the graph G with weights $w_{ij} = -\log r_{ij}$. Because these weights can be negative, we apply the Bellman-Ford algorithm for shortest paths to the graph , taking s as origin.

b) Just iterate the updating procedure once more after $|E||V|$ rounds. If any distance is updated, a negative cycle is guaranteed to exist, i.e. a cycle with $\sum_{j=1}^{k-1}(-\log r_{i_j,i_{j+1}}) < 0$, which implies $\prod_{j=1}^{k-1} r_{i_j,i_{j+1}} > 1$, as required.

5. (20 pts.) **Faster All-Pairs Shortest Path.**

(a) The total weight of a path is the sum of the weights of its edges. When we expand the reweighting formula, the intermediate h values cancel each other out. For example, edge (v_2, v_3) has $-h(v_3)$, but the next edge (v_3, v_4) has $+h(v_3)$. The only h values missing a pair to cancel with are $+h(v_0)$ and $-h(v_k)$:

$$\sum_{i=0}^{k-1} \hat{w}(v_i, v_{i+1}) = \sum_{i=0}^{k-1} (w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1})) = h(v_0) - h(v_k) + \sum_{i=0}^{k-1} w(v_i, v_{i+1})$$

We are left with the sum of the original weights, which is the total weight of the original path, plus $h(v_0) - h(v_k)$. Note that any path from v_0 to v_k shares those endpoints, so $h(v_0) - h(v_k)$ is constant and the total weights of every path maintain their relative ordering. Therefore, the original shortest path is still the shortest path after reweighting.

(b) This part is similar to (a). The total weight of a cycle is the sum of the weights of its edges. The only difference is that here, the starting and ending points are equal, so $h(v_0) = h(v_k)$. This means that every h value will cancel.

$$\sum_{i=0}^{k-1} \hat{w}(v_i, v_{i+1}) = \sum_{i=0}^{k-1} (w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1})) = \sum_{i=0}^{k-1} w(v_i, v_{i+1})$$

Reweighting preserves the exact total weight of every cycle. Therefore, negative reweighted cycles can occur if and only if the original cycle was also negative.

(c) The central observation is that because h is defined using shortest paths from s, the shortest path from s to v must at most equal in length to the shortest path from s to u plus the distance between u and v. This is true because either the path through u is the shortest path to v, in which case they are equal, or the path through u is not the shortest path to v, in which case the shortest path to v must be even shorter. This inequality can be formalized as $h(v) \leq h(u) + w(u,v)$. Subtracting $h(v)$ gives $w(u,v) + h(u) - h(v) \geq 0$. This is exactly our formula for $\hat{w}(u,v)$, so $\hat{w}(u,v) \geq 0$.

(d) We are comparing Floyd-Warshall's $O(|V|^3)$ to this algorithm's $O(|V|^2 \log |V| + |V||E|)$. $|V|^2 \log |V|$ is always preferable to $|V|^3$; the problem is $|V||E|$. In a graph with $O(|V|^2)$ edges (such as a complete graph), $O(|V||E|) = O(|V|^3)$. So, the condition for this algorithm to be faster is that the number of edges must not be $\Omega(|V|^2)$. For example, in a tree, there are $O(|V|)$ edges and Johnson's algorithm is dominated by $O(|V|^2 \log |V|)$, which is much better than $O(|V|^3)$. In short, we say that Johnson's Algorithm is preferable for sparse graphs.

6. (0 pts.) Acknowledgments

(a) I did not work in a group.
(b) I did not consult with anyone other than my group members.
(c) I did not consult any non-class materials.

Rubric:

Problem 1, 20 pts

(a) 8 points: 2 points for each correctly ordered vertex (except s as it's specified as the starting vertex).

(b) 12 points
4 points: correctly identify that the goal is $distance(s,c) < distance(s,d) < distance(s,b)$
2 points: correctly calculate $distance(s,c)$ and $distance(s,b)$
4 points: correctly recognize the two possible cases for $distance(s,d)$
2 points: perform correct calculations for the inequalities with respect to these two cases and obtain the correct final result

Problem 2, 20 pts

(a) 15 pts for correct algorithm

(b) 3 pts for running time analysis

(c) 2 pts for showing the correctness of efficient algorithm

Problem 3, 20 pts

- 5 pts for Dijkstra's algorithm works
- 15 pts for the right proof
  - 5pts Descript what is the shortest path from s to one vertex, let us say v
  - 5pts Use contradiction to suppose another shortest path for this this v
  - 5pts Proof that other path is a contradiction
  - or other right proof

Problem 4, 20 pts

(a) 12 pts:
  - 6 pts for reducing the problem to a shortest path problem.
  - 6 pts for solving the problem with the Bellman-Ford algorithm, and showing why it works

(b) 8 pts:
  - 4 pts for relating the presence of anomaly to a negative cycle in the graph
  - 4 pts for showing how we can find negative cycle efficiently ($O(|V||E|)$, such as with the Bellman-Ford Algorithm)

Problem 5, 20 pts

(a) 6 pts for showing all internal h values cancel, leaving only $h(v_0)$ and $h(v_k)$, which are constant for all possible paths.

(b) 4 pts for showing all h values cancel in a cycle, preserving the original weight.

(c) 6 pts for using the triangle inequality to show the reweight formula cannot be negative.

(d) 4 pts for explaining the given algorithm works better for small $|E|$